Högskolan på Gotland
Passion & vetenskap

Ett Oskarpt Beslut

Om Oskarp Logik i Speldesign

Mikael Hedenström

Examensarbete i speldesign, 15 högskolepoäng
Speldesign och grafik/Speldesign och programmering, vt 2013
Handledare: Kim Solin, Tommi Lipponen
Examinator: Mikael Fridenfalk

Abstract

This paper examines possible applications of fuzzy logic on the field of artificial intelligence in digital games. It describes fuzzy logic in contrast to crisp logic. The paper will give several examples of possible applications of fuzzy logic-based decision making for game AI. These examples include the choosing of weapons for AI agents and the gas-break decision of an AI-controlled car. The paper will also analyze the impacts of game AI which makes decisions based on many factors. It will detail how transitions between AI states can be made more fluid with fuzzy logic. The paper comes to the conclusion that game AI with decision making based on fuzzy logic and fuzzy sets has several advantages and disadvantages. These advantages include the ability for AI agents to make decisions based on many factors. Disadvantages include the increased complexity inherent in poorly constructed fuzzy logic based AI. This complexity emerges when too many variables are part of the rule description, and each variable has several different states. In these cases, an unmanageable amount of rules will be produced.

Sammanfattning

Denna uppsats kommer utforska möjliga applikationer av fuzzy logic på området artificiell intelligens i digitala spel. Fuzzy logic kommer beskrivas i kontrast till skarp logik. I uppsatsen kommer flera exempel ges på möjliga applikationer för beslutsfattande baserat på fuzzy logic inom spel-AI. Bland dessa exempel återfinns val av vapen för AI-agenter samt gas-bromsbeslut hos en AI-kontrollerad bil. Uppsatsen kommer även behandla genomslaget hos en AI som fattar beslut baserade på många faktorer. Den kommer även behandla hur övergången mellan olika AI-tillstånd kan mer flytande med fuzzy logic. Uppsatsen kommer till slutsatsen att spel-AI med beslutsfattande baserat på fuzzy logic och suddiga mängder har flera för- och nackdelar. Dessa fördelar inkluderar möjligheten för AI-agenter att ta beslut baserade på flera faktorer. Nackdelar inkluderar den ökade komplexiteten som uppstår i dåligt konstruerad AI baserad på fuzzy logic. Denna komplexitet uppkommer när regelbeskrivningen innehåller för många variabler och varje variabel har flera olika tillstånd. I dessa fall kommer en ohanterlig mängd regler att produceras.

Innehållsförteckning

Författaren studerar speldesign och programmering vid Högskolan På Gotland. Uppsatsen är ett examensarbete för kandidatexamen inom ämnet speldesign omfattande 15 högskolepoäng och bedrivs på helfart.

Jag vill tacka alla som har varit med och stötta mig under arbetet och deltagarna i min seminariegrupp för värdefulla möten med bra diskussion. Ett stort tack riktas till Kim Solin och Tommi Lipponen, som båda har varit ett väldigt stort stöd under arbetet. Jag vill också rikta ett speciellt tack till Michaela Sundberg som har hjälp mig med synpunkter och varit ett väldigt stort stöd under hela processen. Till sist vill jag tacka Johan Sjöström, som korrekturläst och varit till stor hjälp.

1 Introduktion

1.1 Bakgrund

1.1.1 Speldesign

Människan har i flera tusen år avnjutit spel under olika förutsättningar. Utgrävare har hittat spel i kung Tut´s grav vilket gör att forskarna kan bekräfta att redan de gamla egyptierna spelade spel. Ett av de äldsta spel som mänskligheten känner till anses vara *The Royal Game of Ur*, vilket upptäcktes i en grav runt år 3100 - 2500 f.Kr. Spelet återfanns komplett med instruktioner i en grav i staden *Ur*. Efter att de första brädspelen skapades skulle det dröja flera tusen år innan den första motsvarigheten skapades i USA. 1822 producerades ett av det första amerikanska brädspel som vi känner igen vid namn *Travelers' Tour Through the United States.* Innan dess har många spel kommit och gått, vilket gjort att många av spelen glömts bort under tiden utvecklingen fortskridit (Lee Mitchell 2012:10-14).

1962 släpptes det första spelet vid namnet *Space Wars* som var designat att spelas på datorer av två personer (Schwab 2009:4). Själva spelet gick ut på att skjuta ner den andra spelaren och samtidigt undvika att kollidera med stjärnor. År 1972 släpptes den första maskinen som gjorde det möjligt för människor att spela digitala spel via tv:n hemma. Denna maskin kallades Odyssey och var en liten låda som anslöts direkt till tv:n. Odyssey blev föregångare till modernare konsoler, så som Microsoft Xbox och Nintendo Gamecube. Digitala spel för hemmabruk var länge stationära, men 1989 lanserade Nintendo ett handhållet system som kallades för Game Boy. I dag finns det både konsoler som kopplas till tv:n och handburna system (Lee Mitchell 2012:18-22).

Människan har designat spel utanför akademin i många tusen år. Jobbet för en speldesigner innefattar en process av olika delar. Den första delen för designern är att föreställa sig ett spel och kunna definiera hur spelet fungerar. En stor del är att förklara och göra en inventering över vad som utgör spelet, vilka funktioner som skall finnas etc. Till sist när hela spelet har blivit definierat och alla resurser som behövs finns beskrivna, kan designern behöva förmedla den här informationen till resten av utvecklingsgruppen om en sådan finns. Inom speldesign finns inget definitiv väg som alltid fungerar i utvecklingen av digitala spel och olika spel och idéer kräver olika sätt att tackla problemet på. Till hjälp har en designer olika sorters metoder, tekniker och ramverk att använda sig utav. Det går inte att designa ett framgångsrikt spel genom enbart tur, utan faktorer utanför och inom själva designen på produkten spelar en stor roll (Rollings & Adams 2003:2-5).

1.1.2 Oskarp Logik

I matematikens finns en gren som kallas för logik. Inom logiken studeras påståenden som är exakt formulerade och vattentäta, det skall inte finnas något tvivel om vad ett påstående menar. Ett påstående inom logiken bygger på att det är antigen sant/falskt eller 1/0 (Eriksson & Gavel 2010:179-181). Den här typen av logik brukar kallas för skarp logik (på engelska *crisp logic*). Inom oskarp logik (på engelska *fuzzy logic*) behöver ett påstående inte vara binärt. De kan istället vara diffusa, vilket innebär att påståendet inte enbart behöver behandla sant eller falskt. Acharjya ger i sin bok *Fundamental Approach to Discrete Mathematics* ett exempel på skillnaden mellan skarp och oskarp logik: inom skarp logik kan påståendet "solen går upp i öst" antingen vara sant eller falskt, beroende på vart solen går upp. Men inom oskarp logik kan påståendet "John är en trevlig fotbollsspelare" undersökas. Påståendet graderas då med ett sanningsvärde mellan noll och ett, exempel 0,6 (Acharjya 2009:349-350).

Konceptet med oskarpa mängder (på engelska *fuzzy sets*) lanserades i mitten på 1960-talet av professorn Lotfi A. Zadeh. Han använde sig av oskarp logik för ungefärliga resonemang. Det grundläggande jobbet som gjordes inom oskarp logik och oskarpa mängder under mitten på 1960-talet utgjordes av Zadeh, som anses vara den tidens pionjär inom området (De Silva 2011:611-615). Ett stort antal publikationer är baserade på teorin om oskarpa mängder (på engelska *fuzzy set theory*). Teorin bakom oskarpa mängder har använts inom många olika forskningsområden, som matematik, teknologi och datavetenskap. Oskarp logik har använts under dom senaste 40 åren efter det att Zadeh lanserade konceptet i mitten på 1960-talet (Ma & Chen & Xu 2006:628).

Så sent som under 1970-talet fortsatte utvecklingen inom oskarp logik och olika tillämplingar började göras inom den industriella sektorn i Europa. Logiken användes i utvecklingen av apparater för konsumenter, samt för olika marktransporter och industriella produkter. Den 19 oktober 1987 började elektronikföretaget Toshiba med utvecklingen av AI-system, som kontrollera både maskiner och diverse verktyg. Grunden i de här AI-systemen utgörs av oskarp logik. Själva systemet har kontroll över olika regler, simulationer och värderingar. Toshiba använde sig av oskarp logik på ett praktiskt sätt, och applicerade denna i olika former för industriella produkter. Exempel på industriella produkter är trafikljus och nukleär energi. Under tiden som Toshiba tillverkade AI-systemen, hade Cambridge University utvecklat liknande applikationer baserade på oskarp logik (De Silva 2011:611-615).

1.1.3 Tidigare forskning

Artificiell intelligens är ett väldigt stort och brett område, som innefattar en rad olika tekniker och metodiker. Författaren Brian Schwab behandlar tekniker och metodiker i sin bok *AI Game Engine Programming* som möjliggör att en datorstyrd enhet kan agera. En enhet kan utföra olika typer av uppgifter som bland annat innefattar att samla in data ifrån omvärlden, ta egna beslut och lära sig saker. Fokuset ligger på själva beslutstagandet som baseras på oskarp logik och tekniken kan används inom andra områden utanför spelvärlden.

Det finns mycket forskning kring beslutstagande som baseras på oskarp logik. All forskning sker inte inom spel eller gällande hur enheter i digitala spel kan utföra uppgifter. Nedan följer fyra exempel på olika typer av forskning som baserar sitt beslutstagande på oskarp logik. Den här tekniken används självfallet inom annan forskning också än de exempel som behandlas här.

Två forskare ifrån Korea har baserat ett taktiskt beslutssystem på oskarp logik. De används för undervattensfarkoster, system som skall kunna undvika både torpeder och attacker. För att kunna test hur ubåtar kan undvika torpeder eller andra hot under vatten används en digital simulation (Myeong-Jo & Tae-wan 2012). Dessa beslut är ofta relevanta i spel. Om det finns en datorstyrd enhet som kontrollerar en båt och en spelare som utför attacker, så blir egenskaperna som forskarna tar upp relevanta och applicerbara för ett digitalt spel. Studier har gjort gällande beslutsdelen, som grundar sig i oskarp logik.

Kose undersökte 2012 hur oskarp logik kan användas för att simulera mänskliga beteenden. En del av det här arbetet var att skapa ett väldigt grundläggande spel som påvisar hur beslut kan användas inom digitala spel. Han visade att en datorstyrd enhet kan agera på egen hand utifrån miljömässiga faktorer med hjälp av oskarp logik. Han hävdar att ett system som baserat på oskarp logik kräver klart definierade regler som är korrekt kopplade till den datorkontrollerade karaktären (Kose 2012).

Shaout, King och Reisner har applicerat oskarp logik på det befintliga spelet *Pac-Man*. De tre kom fram till att de, genom att låta spökerna i spelet fatta beslut med hjälp av oskarp logik, kunde öka spökernas upplevda intelligens jämfört med originalspelet. Resultatet ledde till att det blev lättare att lägga till flera regler för spökerna, så att spelaren kunde uppleva att spökena var intelligenta (Shout & King & Reisner 2006).

1.2 Definitioner

1.2.1 Spel

Inom speldesign finns det många olika definitioner på vad ett spel faktiskt är. Eftersom det finns många olika definitioner, kommer den här uppsatsen att använda sig av den som Ernest Adams ger i sin bok *Game Mechanics: Advanced Game Design* och den är endast en utav många definitioner:

> ‘A game is a type of play activity, conducted in the context of a pretended reality, in which the participants try to achive at least one arbitrary, nontrivial goal by acting in accordance with rules’ (Adams & Dormans 2012:1)

Den svenska översättningen till definitionen är följande: *Ett spel är en typ av lekfull aktivitet, utförd i kontexten av en påhittad verklighet, i vilken deltagarna försöker att uppnå minst ett arbiträrt icke trivialt mål genom att handla i enlighet med regler.*

1.2.2 Artificiell Intelligens för spel

Inom artificiell intelligens saknas det en exakt definition av AI (Nilsson 2010:13). Förkortningen AI syftar enbart till spel-AI och inte traditionell AI. Definitionen som används för spel-AI i den här uppsatsen är hämtad ur boken *AI Game Engine Programming* som är skriven av Brian Schwab:

> ‘AI is the creation of computer programs that emulate acting and thinking like a human, as well as acting and thinking rationally’ (Schwab 2009:2).

Den svenska översättningen till definitionen är följande: *AI är skapandet av ett datorprogram som simulerar en människas handlande och tänkande, och dessutom handlar och tänker rationellt*.

Inom spel så behövs egentligen inte den här breda definitionen på vad spel-AI är för någonting. Det viktigaste för spelindustrin är koden som instruerar datorn om hur smarta beslut tas i rätt situation, där det märkbara resultatet för spelaren har en betydande roll.

1.3 Problemformulering

Under de senaste åren har digitala spel utvecklas enormt mycket och spelen har blivit mycket mera realistiska än vad dem var för tjugo år sedan. Utvecklingen har mest innefattat en förbättring av grafiken och ljudet, som båda har utvecklats avsevärt. Men ett problem kvarstår som spelare märker och det är hur AI-utvecklingen har gått. Utvecklingen inom AI har inte nått samma nivå av realism som grafiken. Idag bygger många spel på en icke anpassningsbar AI, vilket innebär att AI i digitala spel inte kan anpassa sig efter spelaren, dvs. lära sig av sina egna misstag. I digitala spel kan inte AI anpassa sig efter spelaren, dvs. lära sig av sina egna misstag. När en spelare överlistar AI:n så kan den utnyttjas och i förläggningen så tröttnar spelaren på spelet (Bakkes & Spronck & Van den Herik 2009).

Många av dagens digitala spel bygger på enklare AI-tekniker. Inom beslutstaganden så är tekniker som state machine och beslutsträd vanligt förekommande. De här två teknikerna är lätta och snabba att implementera, men har sina svagheter. På senare år senare har intresset för bättre tekniker för beslutstaganden ökat, exempelvis fuzzy logic. Tyvärr har inte utvecklarna börjat anamma den här tekniken ännu. Med hjälp av den här tekniken så blir inte övergångarna mellan tillstånd för en AI helt självklara och tydliga som med enklare tekniker (Millington 2009:371 & 293).

1.4 Syfte och frågeställning

Syftet med uppsatsen var att sätta mig in i ämnet oskarp logik och skapa mig en förståelse över hur logiken kan appliceras inom artificiell intelligens, som en beslutsmetod. Jag vill även undersöka vilka fördelar oskarp logik kan tänkas ge för beslutsfattningen inom digitala spel.

Frågeställningen är följande:

- Hur skulle oskarp logik kunna appliceras rent praktiskt inom spel?
- Hur kan oskarp logik appliceras inom AI som en beslutsmetod?
- Vilka fördelar kan oskarp logik tänkas ge som beslutmetod i spel?

2 Metod

2.1 Skarp logik

2.1.1 Matematisk mängd

En mängd inom matematiken är en uppsättning av godtyckliga objekt som är väldefinierade. Ett objekt räknas som väldefinierat när det är distinkt och urskiljbart. Om objektet tillhör en mängd så kallas det för element. Inom mängdläran så spelar ordningen av de ingående elementen ingen vital roll för en mängd (Eriksson 2010:18-20).

Låt oss säga att vi har två olika mängder A och B. Mängden A innehåller fem stycken element som består av alla tal ifrån 1 till 5. Den andra mängden B innehåller tre stycken element med följande namn: Boll, Bil och Båt. Inom mängdläran kan dom båda mängderna A och B presenteras via ett diagram eller en uppräkning av alla ingående element. Nedan följer en mattematisk uppräkning av mängderna A och B (Eriksson 2010:18).

$$A = \{1, 2, 3, 4, 5\} \text{ och } B = \{\text{Boll, Bil, Båt}\}$$

Som det nämndes tidigare spelar ordningen för elementen i en mängd inte någon som helst roll. Därför kan en mängd räknas upp på vilket sätt som helst, ordningen kommer inte att förändra mängden. Exemplet visar att ordningen på elementen inte har någon betydelse för mängderna A och B:

$$A = \{1, 2, 3, 4, 5\} = \{5, 4, 3, 2, 1\}$$

$$B = \{\text{Boll, Bil, Båt}\} = \{\text{Båt, Bil, Boll}\}$$

2.1.2 Operationer för Skarpa Mängder

Inom matematikens mängdlära finns det tre grundläggande operationer som kan användas för att beskriva relationen mellan mängder. De tre vanligaste operationerna är union, snitt och komplement. För att kunna beskriva både operationer och begrepp inom mängdläran, har särskilda matematiska symboler utvecklats (Eriksson 2010:17-23).

Matematiska symbolen för union är ∪. Anta att det finns två mängder A och B. Unionen mellan mängderna (A ∪ B) definieras som en mängd där alla element antingen finns i mängden A eller B, men även dom element som finns i båda mängderna (Koshy 2003:78). Exemplet visar hur en union mellan två mängder A och B ser ut:

$$A = \{1, 2, 3, 4\}, \qquad B = \{2, 4, 5, 6\}$$

$$A \cup B = \{1, 2, 3, 4, 5, 6\}$$

Symbolen för snitt inom mängdläran är ∩. Snittet mellan mängderna A och B (A ∩ B) är den mängd med alla element som både finns i mängderna A och B (Koshy 2003:78-79). Exemplet visar snittet mellan mängderna A och B:

$$A = \{1, 2, 3, 4\}, \qquad B = \{4, 5, 6, 7\}$$

$$A \cap B = \{4\}$$

Komplementet till mängden A skrivs på följande sätt A^c (Eriksson 2010:22). Symbolen för universum är μ och definieras som mängden av alla tillgängliga element (Eriksson 2010:21). Definitionen av A^c är en mängd av alla element som ingår i den universella mängden μ, men inte ingår i mängden A (Koshy 2003:81). Exemplet visar komplementet för mängden A:

$$A = \{1, 2, 3, 4\}, \qquad \mu = \{1, 2, 3, 4, 5, 6, 7\}$$

$$A^c = \mu - A = \{5, 6, 7\}$$

2.1.3 Satslogik

Inom matematikens logiklära finns ett områdets vid namn satslogik. Satslogik innebär att man arbetar med påståenden eller utsagor. En sats är en utsaga, som antingen består av enkla eller sammansatta uttalanden. Varje uttalande kan enbart vara binärt, dvs. sant eller falskt. Satser består oftast av en mängd olika delsatser som är sammankopplade. Om en sats inte går att dela upp i mindre delsatser, kallas den för atomär. De ord som kopplar ihop satser kallas för konnektiv. Sanningsvärdet för en sammansatt sats beror på sanningshalten hos de ingående delsatserna, dvs. om de är sanna eller falska. Om man vill analysera satsernas sanningsvärde används en sanningstabell. I en sanningstabell tilldelas alla atomära satser alla tänkbara kombinationer av värden, därefter tittar man på vilka konsekvenser det får för helheten. För mera information om sanningstabeller hänvisas läsaren till (Eriksson 2010:179-188). Exempel på vad en sats är:

"Gotland är en ö" (sant) eller "12 är ett primtal" (falskt)

2.1.4 Konnektiv

Nedan beskrivs dem tre vanligaste konnektiv som används inom logikläran. De mest grundläggande sammansättningsorden är *och*, *eller* och *inte.* Nedan följer en beskrivning av de vanligast förekommande konnektiven (Eriksson 2010:179-183).

Följande sats: *Jag är liten och bor hemma* består av två olika delsatser. Den första delsatsen är: *Jag är liten* och det andra är: *bor hemma*. Dessa sammanfogas med ordet *och*. När två satser fogas samman med hjälp av ordet *och* kallas det för konjunktion. Symbolen för konjunktion är: ∧. En konjunktion mellan satserna p och q skrivs på följande sätt: (p∧q) (Koshy 2003:5-6). Exempel:

p: *Jag är liten* , q: *bor hemma,*

p∧q = *"Jag är liten och bor hemma"*

Det finns flera sätt att kombinera satser, istället för att använda en konjunktion. Betrakta följande sats: *Lampan kan lysa eller vara släckt*. Satsen består av två stycken delsatser p och q, som binds tillsammans med ordet *eller*. När två delsatser sammanfogas med hjälp av ordet *eller* kallas det för disjunktion. Symbolen för disjunktion är: ∨. En disjunktion mellan satserna p och q skrivs på följandesätt: (p ∨ q) (Koshy 2003:7 - 8). Exempel:

p: *Lampan kan lysa* , q: *vara släckt*

p ∨ q = *Lampan kan lysa eller vara släckt.*

Betrakta följande påstående: *"Det är inte sant att jag kan franska"*. När ordet: *det är inte sant* läggs på en sats, kallas för att negera och operationen kallas för negation. Det är inte alla satser som det går att använda orden: *det är inte sant* på och det resulterar oftast i att ordet *icke* eller *ej* används vid en negation. Symbolen för negation är: ¬ (Eriksson 2010:183). Negation för satsen p skrivs på följande sätt: ¬p (Koshy 2003:8-9).

2.1.5 Matematiska symboler

Alla matematiska symboler som har blivit beskriva tidigare i uppsatsen listas nedan. Symbolerna används inom och kommer ifrån matematikens mängd- och logiklära. Sammanställningen är till för att visa att både logikläran och mängdläran använder sig av symboler som har liknande betydelse.

Tabell 1: Symboler för mängd- och logiklära

	Och	Eller	Inte
Engelska	AND	OR	NOT
Logiska symboler	∧	∨	¬
Mängd symboler	∩	∪	*c*

Källa: 2.1.2 och 2.1.4

2.1.6 Funktion

En funktion är ett matematiskt sätt att visa förhållandena mellan två storheter, som x- och y-värden (Eriksson 2010:231). De matematiska funktioner som är vanligast att komma i kontakt med kan se ut på följande sätt: $y = f(x)$ (Eriksson 2010:244). Funktion inom matematiken betecknas ofta med bokstaven *f*.

Definition för en funktion:

> 'En funktion *f* från mängden *A* till mängden *B* är en regel som till varje element i *A* (det vill säga varje möjligt *x*-värde) kopplar ett och endast ett element i mängden *B* (det vill säga ett och endast ett *y*-värde)' (Eriksson & Gavel 2010:245).

Definitionen brukas skriva matematiskt på följande sätt: $f : A \rightarrow B$ (Eriksson 2010:245).

Funktioner beskrivs på olika sätt för både inom programmering och matematik. En programmerare som är van vid programmeringsspråket C, kommer att vara van vid en annan typ av representation av funktioner. Vi har följande matematiska funktion:

$$f(x) = x^2$$

Funktionen ovan använder värdet på *x* för att utföra operationen upphöjt till två. Låt oss säga att *x* har värdet 4 och stoppas därefter in i funktionen. Resultatet skulle bli 16, eftersom 4 * 4 = 16. Den här matematiska funktionen skulle kunna representeras i programmeringsspråket C på ett helt annat sätt. Följande exempel visar hur funktionen ovan kan se ut i C:

```
void f(int x)
{
	int n;
	int v;
	n = x;
	v = n;
	while (n > 0)
	{
		x = x + v;
		n = n - 1;
	}
}
```

I funktionen stoppas ett reellt tal *x* som är av typen heltal (int) in. Det första som händer i funktionen är att två heltal *n* och *v* skapas. Därefter så lagras värdet som *x* har i variabeln *n,* och värdet som *n* har lagras i *v*. Det sista som händer är en uppräkning där *x* kommer att adderas med det ursprungliga värdet som skickades in i funktionen, dvs. värdet som lagras i *v*. Den här uppräkningen kommer att fortskrida så länge som värdet på *n* är större än 0, varje varv i loopen subtraheras talet som finns lagrat i *n* med 1. Funktioner representeras på olika sätt inom olika fält, som programmering och matematik.

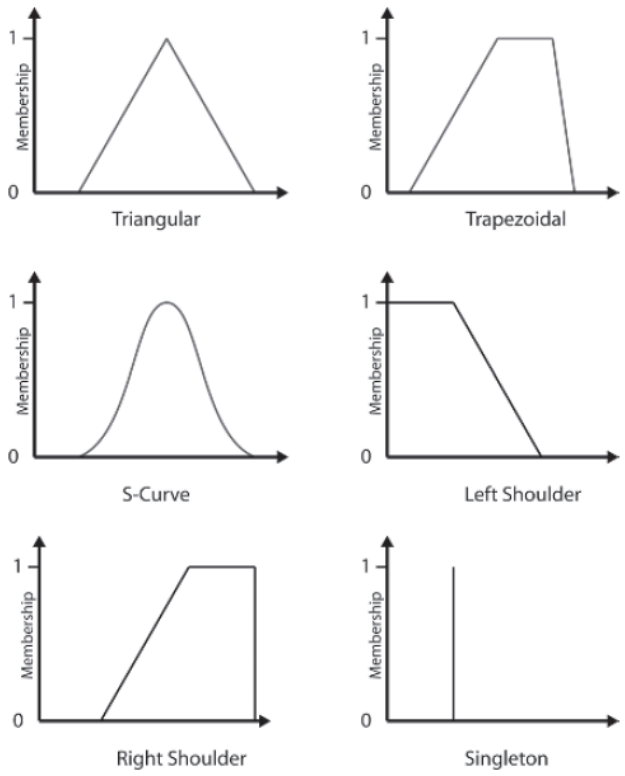
2.2 Oskarp logik

2.2.1 Oskarpa mängder

Inom matematikens mängdlära tillhör ett element antingen en mängd eller inte. Elementet kan aldrig kanske ingå i en mängd, eller endast delvis tillhör en mängd. För oskarpa mängder finns inte den här tydliga gränsen att ingå eller vara utanför mängden. Här pratas det istället om vilket medlemskap eller vilken grad av tillhörighet ett element har till mängden. Inom skarpa mängder så kan en person antingen vara hungrig eller inte, dvs. 0 för hungrig och 1 för mätt. För oskarpa mängder finns inte den här binära skillnaden, istället kan en oskarp mängd ta alla värden i intervallet [0,1]. En uppräkning sker inte på samma sätt som för en skarp mängd, istället används funktioner för att beskriva en oskarp mängd (Demico & Klir 2003:12-13).

2.2.2 Medlemsfunktion

En medlemsfunktion är en matematisk funktion som används för att beskriva och definiera en oskarp mängd. Funktionen kan anta olika former och de vanligaste är triangulär- och trapetsform. Man kan se på graferna i figur 1 att de definierar en gradvis övergång ifrån ett område som ligger i mängden till ett annat området som befinner sig helt utanför mängden, vilket möjliggör att ett värde kan ha ett delvis medlemskap till mängden. Det här är kärnan i oskarp logik. En medlemsfunktion kan skrivas matematiskt på följande sätt: $F_{mängdens\_namn}(x)$. Nedan visas olika typer av medlemsfunktioner (Buckland 2005:419-421).



Figur 1: Medlemsfunktioner (Buckland 2005:420)

Rent matematiskt inom oskarp logik går det att definiera en medlemsfunktion på ett annat sätt än vad som angivits tidigare. Det finns en oskarp mängd A och den här oskarpa mängden A används också för att beteckna en medlemsfunktion. Trots att symbolen A utnyttjas för två olika definitioner så råder det inga oklarheter, eftersom en oskarp mängd är fullständig och unikt definierad av en särskild utvald medlemsfunktion (Demico 2003:13). Exempel på hur en oskarp medlemsfunktion betecknas:

$$A: X \rightarrow [0,1]$$

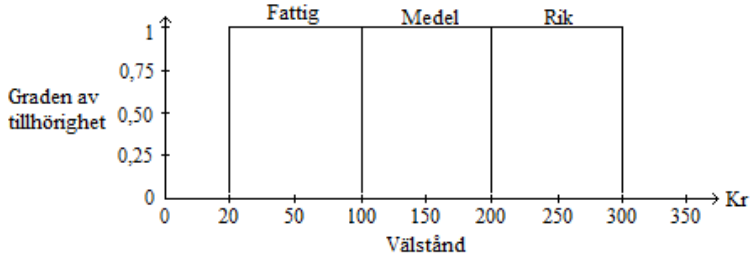
Värdet för *A(x)* uttrycker graden av medlemskap för elementet x i den oskarpa mängden *A* (Demico 2003:13).

En medlemsfunktion beskriver en oskarp mängd, med hjälp av olika formationer för funktionen som visas i figur 1. I följande exempel undersöker vi universumet av välstånd, som definieras av mängderna *fattig*, *medel* och *rik*. De tre olika mängderna representerar hur mycket pengar det behövs för att tillhöra en viss kategori. Värdena för mängderna är fiktiva och inte modellerade efter etablerade definitioner av välstånd. Vi börjar med att definiera de här tre mängderna som skarpa mängder. Detta representeras grafiskt i figur 2. Därefter presenteras samma exempel fast med oskarp logik, vilket visas i figur 3. Figuren använder samma förutsättningar.

$$Fattig = \{ 20, 21, 22, 23, \dots , 99 \}$$

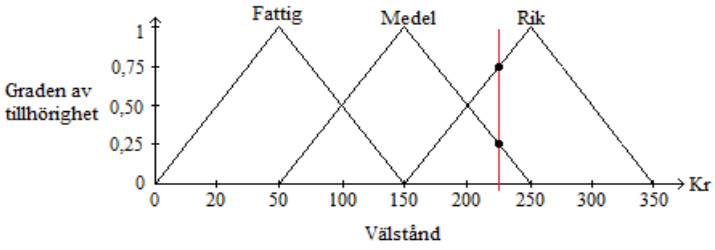
$$Medel = \{ 100, 101, 102, 103, \dots , 199 \}$$

$$Rik = \{ 200, 201, 202, 203, \dots , 299 \}$$



Figur 2: Representation av skarpa mängder

Figur 2 ger en bild över intervallen och vilka mängder som utgör välstånd i det här exemplet. För att en person skall kunna vara rik, krävs det att denne har minst 201 kr. En person som har 199 kr kommer att kategoriseras som medelmåttig, medan många skulle kategorisera denne som rik. Problemet med att skarpa mängder är när personer befinner sig väldigt nära gränsvärden. Om en person har 99 kr och en annan 101 kr, så är de kategoriserade som fattig respektive medelrik. Deras inkomst är ganska lika. Skarpa mängder ger de här skarpa övergångarna mellan mängderna. Fördelen med oskarpa mängder är att skillnaderna mellan mängderna inte blir lika skarpa. Figur 3 visar samma exempel med oskarpa mängder.



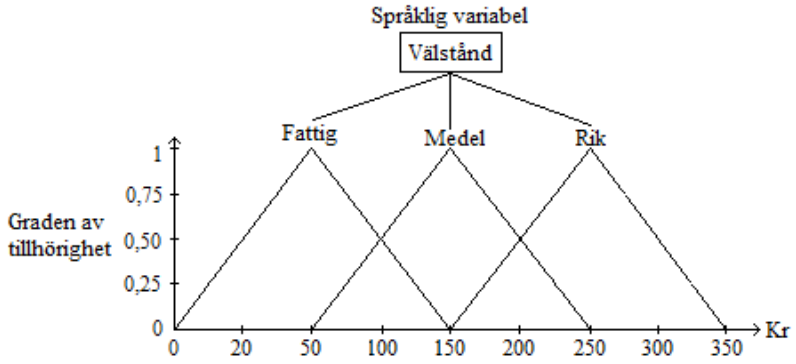
Figur 3: Representation av oskarpa mängder

Medlemsfunktioner definierar och beskriver en oskarp mängd. Figur 3 visar att mängderna går ihop och att skillnaderna inte blir lika tydliga. De båda punkterna som skärs av den röda linjen ger en del information. I det här fallet skulle en person vara rikare än en fattig person, men fattigare än en rik. Det går att påstå att personen tillhör den oskarpa mängden rik till 0.75 och medel 0.25. Oskarpa mängder baseras på graden av tillhörighet och inte på om ett element tillhör mängden eller ej.

2.2.3 Oskarpa språkliga variabler

När variablers värden består av ord eller meningar på naturliga eller konstgjorda språk kallas de språkliga variabler (Bojadziev & Bojadziev 2007:45). Språkliga variabler är lättare för en människa att förstå och erbjuder en mer sofistikerad representation av ett värde. Fördelen med oskarpa mängder är att man inte behöver vara begränsad till precisionen för skarpa värden, dvs. heltal (Champandard 2003:261). Oskarpa språkliga variabler är en sammansättning av en eller flera oskarpa mängder som kan representera ett koncept. Låt säga att det finns tre stycken oskarpa mängder: *fattig*, *medel* och *rik* som tillhör den oskarpa språkliga variabeln *välstånd* (Buckland 2005:419). Välstånd är en språklig variabel som representerar värden med hjälp av ord som *fattig*, *medel* och *rik*. De här värdena kallas för villkor för den språkliga variabeln välstånd och utryckts av oskarpa mängder. Bojadziev ger denna förklaring, men variabeln ålder är utbytt mot välstånd som visas i figur 3 (Bojadziev 2007:44-45). Exemplet nedan visar att den språkliga variabeln är uppbyggd av tre oskarpa mängder.

Välstånd = {Fattig, Medel, Rik}



Figur 4: Visar de oskarpa mängderna som utgör den språkliga variabeln.

Figur 4 visar att den språkliga variabeln är uppbyggd av tre villkor, som utgörs av oskarpa mängder. På det här sätet kan konceptet välstånd utryckas på ett relativt lätt sätt.

2.2.4 Oskarpa regler

Oskarpa regler relaterar till ett känt medlemskap för en given oskarp mängd som sedan kan medlemsvärden för andra oskarpa mängder (Millington 2009:380). En oskarp regel byggs upp av först ett påstående (försats) och sedan en konsekvens (eftersats). Det första som händer i en oskarp regel är att ett påstående beskrivs och sedan om påstående stämmer, används konsekvensen. En konsekvens kan vara alltifrån en ny beräkning till att olika värden sätts för variabler eller mängder. Konsekvensen i en oskarp regel kan lösas ut igenom graden av tillhörighet. Följande form visar hur en oskarp regel byggs upp och beskrivningen är hämtad från (Buckland 2005:424-425).

IF påstående THEN konsekvens

Några exempel följer på oskarpa regler:

IF Syns fienden THEN vänd båtens för emot fienden

IF Är fienden nära AND ammunitionen inte är slut THEN Skjut

IF Är skeppet träffat THEN Ta med utrustning AND Sätt dig i en livbåt

Ett påstående kan utgöras av en eller flera oskarpa villkor. Graden av medlemskap för ett påstående definierar vilken grad konsekvensen kommer att utgöras av. De här reglerna byggs upp av skarpa villkor, som utgörs av skarpa mängder (Buckland 2005:425). Skulle reglerna utgöras av en skarp mängd, då skulle reglerna bli mera binära. Exempel vis så den här regeln enbart lösas ut om fienden syns helt, eftersom antingen syns båten eller inte.

IF Syns fienden THEN vänd båtens för emot fienden

Om regeln utgörs av oskarpa villkor eller mängder så kommer regeln inte enbart utföras om hela båten syns. Nu kommer inte regeln att vara binär, utan kommer att aktiveras om graden av tillhörighet är tillräcklig hög. Fienden behöver inte synas då värdet är ett, utan kan även synas när värdet är 0.7 eftersom delar av båten är dold. Den här typen av regler kan utgöras av både skarpa och oskarpa villkor, men ger en stor skillnad i resultatet.

2.2.5 Oskarpa operatorer

Konceptet för oskarp logik liknar det som används inom oskarp mängdteori. Många av symbolerna som togs upp inom mängdläran och satslogiken har bytts ut till andra symboler som används inom oskarp logik. Nya operatorer som används inom oskarp logik kommer att listas nedan. Enbart motsvarigheter till de symboler som vi tidigare har gått igenom kommer att listas. Listan är hämtad ifrån (Champandard 2003:264).

Tabell 2: Sammanställning av matematiska symboler

Engelska	Logiska Symboler	Mängd symboler	Ekvation
NOT	¬	c	1 - x
AND	∧	∩	MIN(x, y)
OR	∨	∪	MAX(x, y)

Snittet för en oskarp operation är matematiskt ekvivalent med AND-operatorn. Resultatet som blir när två oskarpa mängder använder operatorn AND, är en ny oskarp mängd (Buckland 2005:421). Varje oskarp operation har en numeriskt regel som tillåter beräkningar gällande graden av sanning. Dessa baseras på graden av sanning för varje oskarp mängd som används i en oskarp funktion. I följande matematiska beskrivning är m_A och m_B graden av tillhörighet av de oskarpa mängderna *A* respektive *B*. Nedan följer ett exempel på hur AND-operatorn används inom oskarp logik (Millington 2009:379).

$$m_{(A \text{ AND } B)} = \min(m_A, m_B)$$

Union för en oskarp mängd är mattematisk ekvivalent med OR-operatorn. När man använder operatorn OR, bildas en ny oskarp mängd. Den här mängden är resultat av den största graden av tillhörighet från två andra mängder (Buckland 2005:422). OR-operatorn fungerar på samma sätt som AND fast istället för att söka efter det minsta värdet, söks det största. Enligt författarna Millington och Funge skrivs OR ut matematiskt på följande att som visas nedan. m_A och m_B är graden av tillhörighet av mängderna *A* respektive *B* (Millington 2009:380).

$$m_{(A \text{ OR } B)} = \max(m_A, m_B)$$

Komplementet för ett värde som representerar graden av tillhörighet för en given mängd är ekvivalent med operatorn NOT. (Buckland 2005:422). Precis som inom traditionell logik, NOT-operatorn relaterar endast till ett enda faktum, medans AND och OR relaterar till två faktum. Den matematiska beskrivningen för NOT operatorn visas nedan. m_A är graden av tillhörighet för mängden *A* (Millington 2009:380).

$$m_{(\text{NOT } A)} = 1 - m_A$$

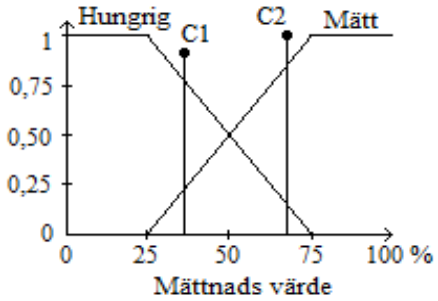
2.3 Artificiell Intelligens

2.3.1 Fuzzification

Fuzzification är en process som skapar oskarpa variabler utifrån ett oskarpt värde och det här sker igenom oskarpa mängder. En medlemsfunktion definierar värdet av en oskarp variabel (Champandard 2003:262). Eftersom de flesta spel inte sparar sin data internt i form av oskarpa mängder, behövs det ett sätt att konvertera data på. När vanlig data i spel omvandlas till graden av tillhörighet, kallas det för *fuzzification* och när data omvandlas tillbaka kallas det för *Defuzzification.* Det finns olika typer av *fuzzification*- metoder som konverterar data på olika sätt (Millington 2009:373).

En av de vanligaste *fuzzification*-teknikerna är att omvandla numeriska värden till medlemskap i en eller flera oskarpa mängder. Den här metoden kallas för *numeric fuzzification* (Millington 2009:373 - 374)

Vi använder oss av ett exempel som är baserat på författarna Millington och Funge exempel över *numeric fuzzification* (Millington 2009:373). Det finns två karaktärer som betecknas C1 och C2. Båda karaktärerna har varsitt numeriskt värde som beskriver hur mätta de är. Värdena omvandlas ifrån ett numerisk värde till en medlemskap för *Hungrig* och *Mätt* som är oskarpa mängder. Det hela sköts igenom en medlemsfunktion. För varje oskarp mängd kommer värdena som skickas in i de funktionerna att omvandlas till grad av medlemskap för en viss mängd. Figuren nedan visar dem båda medlemsfunktionerna för *hungrig* och *mätt*.



Figur 5: Beskriver mättnads värde för karaktärerna

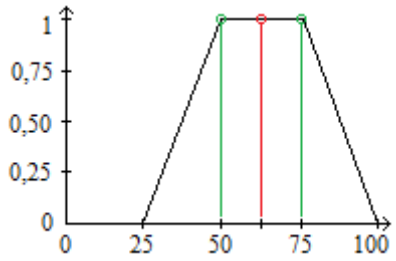
Utifrån den här mängden av funktioner, kan medlemsvärden utläsas. Värden för två karaktärer (C1 och C2) syns i figur 5: C1 är hungrig 0,75 och mätt 0,25, medans C2 är 0,20 hungrig och 0,80 mätt. Värt att notera i det här fallet är att det utgående värdet från de båda medlemsfunktionerna har summan av 1.

Det finns en annan teknik för *fuzzification* och den behandlar andra sorters datatyper än numeriska. Andra värden som booleska och uppräkningar behöver ibland använda sig av *fuzzification* inom spel. Ett vanligt sätt att fördefiniera medlemskapsvärden för varje relevant mängd. Om en spelare håller i ett vapen eller inte så kan en boolesk variabel indikera om vapnet används eller inte med hjälp av värdena 1 eller 0. Samma struktur som för booleska variabler fungerar även för uppräkningar. Skillnaden är att en uppräkning kan representera flera val istället för ett (Millington 2009:374).

2.3.2 Defuzzification

Defuzzification fungerar omvänt än vad *fuzzification* gör. Nu handlar det inte om att ta ett numeriskt värde och göra det till en oskarp mängd, utan att omvandla en oskarp mängd till ett numeriskt tal. Det finns många olika tekniker för att göra den här typen av omvandling (Buckland 2005:432).

Mean of Maximum (Mom) är en teknik för att utföra *defuzzification.* Själva tekniken går ut på att beräkna medelvärdet på de utgående värdena som har högst grad av medlemskap, dvs. högst tillförlitlighet. Figuren nedan visar ett exempel som baseras på författaren Buckland's exempel över *Mom*-tekniken (Buckland 2005:434).



Figur 6: Beskriver medelvärdet

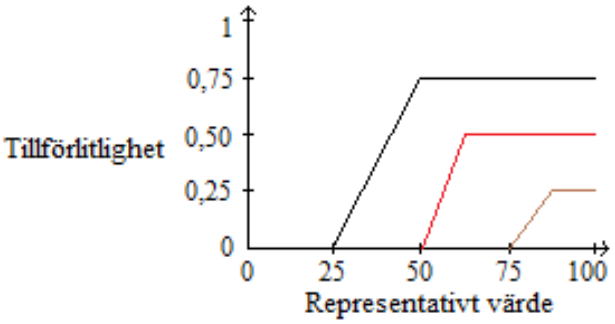
Funktionen i figur 6 har två maxvärden som motsvarar 50 och 75, båda värdarna representeras av de gröna linjerna. Den röda linjen anger medelvärdet mellan de gröna linjerna. Medelvärdet för de gröna punkterna beräknas enligt: (50 + 75) / 2 = 62,5.

Centroid är en annan teknik för att utföra *defuzzification*. Tekniken går ut på att beräkna arean av den yta som en medlemsfunktion täcker. Ett skarpt värde kan motsvara vikten för en hel oskarp mängd, eller medelvärdet för varje area. Det här tillvägagångssättet kräver mycket beräkningskraft, eftersom tekniken förlitar sig till en stor del på analyser eller matematiska kunskaper (Champandard 2003:264).

Average of Maxima(MaxAv) är en annan teknik för att utföra en *defuzzification*. Maximavärdet eller det representativa värdet för en oskarp mängd är de värde som har 1 i medlemskap. För en triangulär mängd är det lätt att hitta det medlemsvärde som har 1, (det är spetsen på triangel eller mittpunkten). För andra mängder med formationer som *right shoulder, left shoulder* eller *trapezoidal*, är medelvärdet det värde som befinner sig mellan början och slutet av platån som bildas (den platta ytan som inte lutar). MaxAv-tekniken skalar det representativa värdet för varje konsekvens med deras tillförlitliga värde. Det skarpa värdet sätts därefter till medelvärdet. Matematiskt ser det ut på följande sätt som visas nedan (Buckland 2005:436).

$$\text{Skarpt värde} = \frac{\sum representativt \quad värde * tillf\ örligtlig\ heten}{\sum tillf\ örliglig\ heten}$$

Figur 7 är baserat på ett exempel där författaren Buckland beskriver *MaxAv*. Det finns tre oskarpa mängder *snabbt* (svart), *medelfart* (röd) och *sakta* (brun) som beskriver den språkliga variabeln hastighet.



Figur 7: Visar språkliga variabeln hastighet

Tabell 3 nedan visar värdena som representeras i figuren ovanför.

Tabell 3: Sammanställning av farterna

Mängder	Representativt värde	Tillförlitlighet
Snabbt	50	0,75
Medelfart	37,5	0,50
Sakta	12,5	0,25

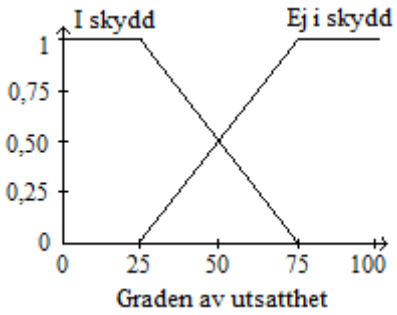
Följande värden beräknas enligt formeln ovan:

$$\text{Skarpt värde} = \frac{50*0,75+37,5*0,5+12,5*0,25}{0,75+0,5+0,25} = \frac{59,375}{1.5} \cong 39.6$$

3 Resultat

3.1.1 Tillämpning

Varje beslut har vilken typ som helst av skarp input. Den här typen av input kan antingen vara ett numeriskt värde, en uppräkning eller booleskt värde. Varje input-värde konverteras till oskarpa tillstånd, genom användning av medlemsfunktioner. Medlemsfunktioner har förklarats tidigare i uppsatsen. För en del implementationer i spel. Delas input-värdet till två eller flera oskarpa tillstånd och summan av tillstånden blir 1. Figurerna nedan visar medlemsfunktioner som skapar oskarpa tillstånd för beslutstaganden och baseras på författarna Millington och Funges exempel (Millington 2009:381-382).



Figur 8: Beskriver språkliga variabeln utsatthet

Figur 8 ovan visar att det finns två stycken oskarpa tillstånd: antingen befinner sig objektet i skydd eller så gör den inte det. Skarpa inputvärden konverteras till många olika tillstånd. Alla tillstånd kan arrangeras antingen i olika grupper eller via ömsesidighet till varandra. Det finns både ingående- och utgående tillstånd. De ingående tillstånden är normalt sett oskarpa tillstånd som representerar olika möjliga handlingar som en karaktär kan tänkas utföra. Tidigare har uppsatsen behandlat oskarpa regler och hur de kan konstrueras med hjälp av operatorer. Nedan följer ett exempel som författarna Millington och Funge har konstruerat för att förklara oskarpa regler (Millington 2009:382).

IF jaga AND på väg in i ett hörn AND snabb förflyttning THEN bromsa

IF leder AND framför ett hörn AND seg förflyttning THEN accelerera

Reglerna är strukturerade på det sätt att varje moment är ett tillstånd från en annan skarp input. Klausulerna kombineras alltid med AND-operatorn för oskarp logik. I exemplen som angavs ovan finns det tre stycken klausuler. Att det finns tre beror på att det användes tre olika ingående värden, där varje klausul representerar ett av tillstånden från varje input. Ett vanligt förfarande är att använda en komplett uppsättning regler: en regel för varje möjlig kombination av tillstånd som generas ifrån det ingående värdet. Om vi använder exemplet som angavs tidigare finns 18 (= 2*3*3) (Millington 2009:382).

Nedanstående exempel är baserat på båda författarna Millington och Funge. Målet med exemplet är att visa hur både oskarp logik och oskarpa mängder kan användas för att ta beslut om hastighetsförändring inom ett digitalt spel. För att generera output gås varje regel igenom. Sedan beräknas graden av tillhörighet för det utgående tillståndet. Det här sker på följande sätt: man tar det minsta graden av tillhörighet för tillståndet som går in i regeln. Den slutgiltiga graden av tillhörighet för varje tillstånd är det maximala värdet av alla applicerbara regler. Följande exempel använder sig av två olika påståenden som jämförs. Det genereras endast ett tillstånd från varje regel. Reglerna som kommer att användas är uppbyggda på det här sättet. I verkligheten kan en regel använda sig av flera påståenden, men för att underlätta exemplet används enbart två. Blocken av regler ser ut på följande sätt (Millington 2009:383).

IF kurvan börjar AND snabb förflyttning THEN bromsa

IF kurvan slutar AND snabb förflyttning THEN accelerera

IF kurvan börjar AND långsam förflyttning THEN accelerera

IF kurvan slutar AND långsam förflyttning THEN accelerera

Följande grad av tillhörighet används:

Kurvan börjar: 0.2

Kurvan slutar: 0.8

Snabb förflyttning: 0.5

Långsam förflyttning: 0.7

Resultatet ifrån varje regel är:

Bromsa = min(0.2, 0.5) = 0.2

Accelerera = min(0.8, 0.5) = 0.5

Accelerera = min(0.2, 0.7) = 0.3

Accelerera = min(0.8, 0.7) = 0.7

Till slut blir värdet för att kunna bromsa 0.2, som är ett resultat av den första regeln. Den högsta graden av acceleration för reglerna är 0.7 och ges av den fjärde regeln. Låt oss titta närmre på den andra regeln för acceleration. Det vi vet om den regeln är att output blir åtminstone 0.5. När vi stöter på den tredje regeln som ger resultatet 0.3, då kommer vi kunna dra slutsatsen att den aldrig kommer generera ett värde som är större än 0.3. Värdet för regel nummer två är 0.5. Den här regeln kan inte möjligen vara det maximala värdet för acceleration, vilket gör att vi kan sluta att använda denna regel. Efter det att alla grader av tillhörighet har genererats som utgående värden, kan vi helt enkelt gå vidare med att utföra *defuzzification* för att undersöka vad som skall hända härnäst. Konsekvenserna av reglerna har i exemplet fått ett numeriskt värde som avgör hur hårt en inbromsning eller acceleration skall utföras på (Millington 2009:383).

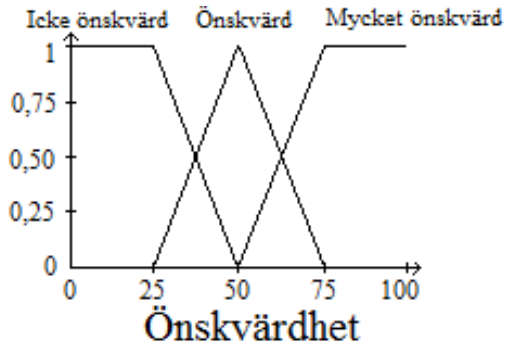
3.1.2 Vapenbyte

Kommande exempel visar hur en spelkaraktär använder oskarp logik för avgöra vilket vapen som skall användas i rätt situation. Hela exemplet är baserat på ett annat exempel som är designat av författaren Buckland i hans bok *Programming Game AI by Exampel*.

Det finns många exempel på hur oskarp logik och språkliga variabler byggs upp på, vilket kan visas upp på ett lätt och pedagogiskt sätt genom att designa ett system som väljer vapen. För att hålla system på en grundläggande nivå så baseras själva valet med hjälp av två olika faktorer: själva distansen till fienden och hur mycket ammunition som finns kvar i vapnet. De språkliga variablerna som kommer att användas i det här exemplet är: distansen till målet, status för ammunitionen och önskvärdhet. Reglerna som styr vilket vapen som skall väljas för en specifikt uppgift, baseras på hur önskvärt ett specifikt vapen är för given situation. Det här möjliggör att en NPC (agent) kan basera sitt val av vapen med hjälp av ett värde som baseras på den högsta önskvärdheten. Både distansen till en fiende och önskvärdhet kommer vara densamma för alla typer av vapen i exemplet, men ammunitionskapaciteten är inte den samma för alla vapensystem. Exemplet fokuserar enbart på design av regler för en racketkastare (Buckland 2005:425-426).

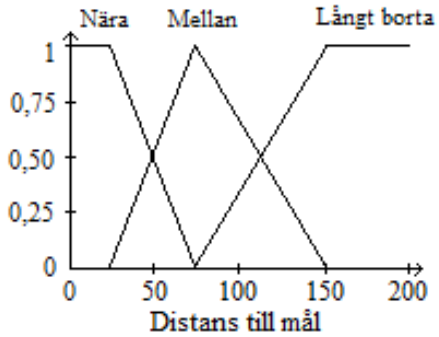
När man designar språkliga variabler finns det en del viktiga regler att förhålla sig till. Det första man bör tänka på när man designar oskarpa språkliga variabler är att beteckna konsekvensmängden, som i det här fallet är önskvärdhet. En vertikal linje (som representerar det ingående värdet) dras igenom den språkliga variabeln. Summan av tillhörighet för varje oskarp mängd som den vertikala linjen skär bör bli ungefär 1. Det här skall göra så att övergångarna mellan värden blir mjukare. En vertikal linje skall helst skära igenom två eller mindre oskarpa mängder (Buckland 2005:425-426).

Värdet för önskvärdheten behöver kunna representera alla heltal mellan 0 och 100. Önskvärdhet kommer att använda sig av tre medlemsmängderna: *icke önskvärd, önskvärd och mycket önskvärd*. Tillsammans måste alla tre mängderna täcka in hela intervallet från 0 till 100. Figuren nedan visar indelningen (Buckland 2005:425-427).



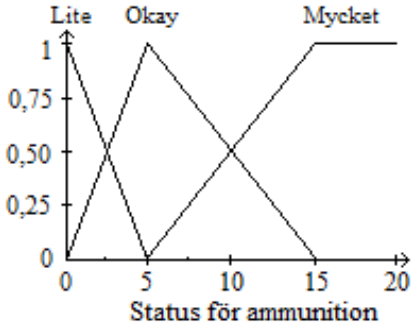
Figur 9: Språkliga variabeln önskvärdheten

Distansen till målet består av tre medlemsmängder och beskriver på vilket avstånd ett mål befinner sig på: *nära, medel* och *långt borta*. När en spelare spelar ett spel så är det vanligt att denne tänker att fienden är nära eller långt borta, och av det skälet har mängderna delats in efter avstånd. Avståndet från NPC till målet mäts i antalet pixlar, där 25 pixlar är nära, 75 pixlar är medelavstånd och långt avstånd är minst 150 pixlar eller mer. Distanserna visas i figuren nedan (Buckland 2005:427).



Figur 10: Beskriver språkliga variabeln distans.

Statusen för hur mycket ammunition som finns kvar består av följande medlemsmängder: *lite ammunition, ammunition ok* och *mycket ammunition*. De här medlemsvariablerna skiljer markant mellan olika vapensystem. En pistol och granatkastare har exempelvis inte samma ammunitionskapacitet. En raketkastare kan skjuta ett skott per sekund. När ammunitionsnivån är ok finns det 5 raketer. Ammunitionen räknas som låg när det finns mindre än 5 och en hög nivå motsvarar 15 raketer. Figuren nedan visar statusen för ammunitionen. (Buckland 2005:428).



Figur 11: Beskriver ammunitions statusen.

Regler måste skapas som beskriver alla tänkbara möjligheter och kombinationer för varje språklig variabler som har beskrivits ovan. Eftersom vi har tre språkliga variabler så behövs det nio olika regler. Enligt författaren Buckland så är en raketkastare väldigt farlig att använda på korta avstånd, då chansen är väldigt hög att skadas av explosionen. En raket flyger relativt sakta och det gör att på långa avstånd är en raketkastare inte förstahandsvalet. Med dessa aspekter i åtanke, har följande regler utformats (Buckland 2005:428-429).

1. IF mål_långt_borta AND ammuntion_mycket THEN önskvärd
2. IF mål_långt_borta AND ammunition_ok THEN icke_önskvärd
3. IF mål_långt_borta AND ammuntion_lite THEN icke_önskvärd
4. IF mål_mellan AND ammuntion_mycket THEN mycket_önskvärd
5. IF mål_mellan AND ammuntion_ok THEN mycket_önskvärd
6. IF mål_mellan AND ammuntion_lite THEN önskvärd
7. IF mål_nära AND ammuntion_mycket THEN icke_önskvärd
8. IF mål_nära AND ammuntion_ok THEN icke_önskvärd
9. IF mål_nära AND ammuntion_lite THEN icke_önskvärd

Slutligen är det dags att studera den oskarpa slutgiltiga processen. Detta sker genom att testa några värden och se vilka regler som används respektive inte används. Vi börjar med att beräkna medlemskapen för de ingående värdena som tillhör en viss regel. Beräkna därefter fram slutsatser som är baserad på beräkningen som gjordes för medlemskapen. Kombinera sedan alla slutsatser till en slutsats (oskarp mängd). Om det är crisp-värden måste slutsatsen gå igenom en *defuzzification*. Varje regel skall beräknas fram på det här sättet (Buckland 2005:429).

Anta att en NPC har 100 pixlar till sitt mål och har 5 raketer kvar. Nedan listas alla grader av tillhörig för alla påståenden som är relevanta för just det här fallet. Sammanställningen gäller enbart när NPC:n har 100 pixlar kvar till sitt mål och har 5 raketer kvar. Informationen återfinns i graferna som har genomgåtts i början av exemplet.

1. mål_långt_borta = 0.35
2. mål_mellan = 0.65
3. mål_nära = 0.0
4. ammuntion_mycket = 0.0
5. ammunition_ok = 1.0
6. ammuntion_låg = 0.0

I tabell 4 görs en sammanställning av alla regler. Då reglerna binds samman av AND-operatorn så används det minsta värdet av två påståenden.

Tabell 4: Sammanställning av reglerna

	Mål nära	Mål mellan	Mål långt borta
Ammunition lite	Icke önskvärd 0.0	önskvärd 0.0	Icke önskvärt 0.0
Ammunition ok	Icke önskvärt 0.0	Mycket önskvärd 0.65	Icke önskvärt 0.35
Ammunition mycket	Icke önskvärt 0.0	Mycket önskvärd 0.0	önskvärd 0.0

När man har en NPC:n som har 100 pixlar till sin fiende och har enbart 5 raketer kvar, visar tabellen ovan vilka regler som kommer att användas. När NPC:n befinner sig på ett mellanavstånd eller lång bort från sitt mål och ammunitionen är ok, kommer endast två regler att kunna användas. Dessvärre så finner NPC:n att det inte är så önskvärt att skjuta sin fiende på ett långt avstånd om det är ont om raketer, men om fienden finns på ett mellanstort avstånd så är chansen väldigt stor att NPC:n skjuter.

4 Analys

4.1 Regler kan medföra nya regler

När man skapar förutsättningar för oskarpa regler, så är det viktig att ha med sig i bakhuvudet att det snabbt blir många regler som skapas och utvecklas. Ju fler ingående mängder som en regel baseras på, desto fler olika regler kommer att behövas. Fördelen med oskarp logik är att det går att göra skillnaderna mellan olika tillstånd väldigt luddiga, vilket bidrar till att en NPC blir svårare att läsa och förstå sig på. Använder man traditionell logik så är det lättare för en spelare att se i vilket tillstånd en NPC befinner sig i, eftersom den antingen är i ett tillstånd eller inte.

Lått säga att det finns två olika regler och varje regel har tre ingående mängder och en utgående. Vi lånar reglerna från Millington och Funge:

IF jagad AND på väg in i en kurva AND snabb förflyttning THEN bromsa

IF leder AND befinner mig i en kurva AND seg förflyttning THEN accelerera

Varje kombination av de ingående mängderna skapar ett möjligt tillstånd för en NPC att befinna sig i. Alla tillstånd har sin egen utgående mängd. Dessa regler kommer en utvecklare att behöva skapa. Om vi beräknar hur många möjliga regler som går att skapa utifrån de här två reglerna, så kommer vi att få resultatet som visas nedan.

Regler att utgå ifrån: 2

Antalet ingående mängder: 3

Antalet kombinationer per mängd: 3

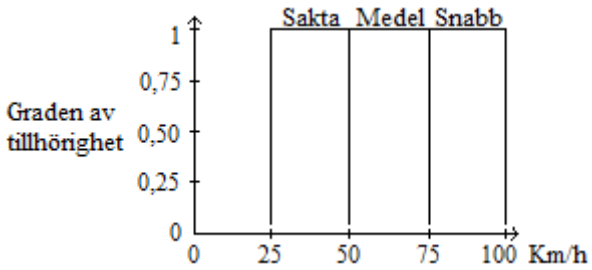
Beräknar ihop alla tre siffror enligt: (2*3*3 =) 18. Dom två reglerna som var angivna från början med tre ingående parametrar skapar 18 möjliga regler. Varje regel måste skapas manuellt eftersom en dator inte vet vad resultatet av en regel skall vara. Det blir snabbt många regler som måste skapas utifrån få mängder. Enligt författarna Millington och Funge gäller följande: om ett system med 10 ingående variabler skulle processeras och varje variabel har 5 tillstånd, skulle ungefär 10 miljoner regler skapas (Millington 2009:387).

Oskarp logik kan vara väldigt kraftfull men skapar snabbt mycket jobb för en utvecklare. Nackdelen är ett system med många ingående värden blir snabbt väldigt stort, till och med för stort. Så länge ett system har få antal värden som ingår i en regel uppstår inte det här problemet. Det kan vara bättre med många system med få ingående värden än ett stort. Det finns metoder som kan skapa regler utifrån givna förutsättningar och det skulle kunna lösa det här problemet (Millington 2009:387).

4.2 Övergångar mellan tillstånd

Det finns olika typer av beslutsmetoder inom artificiell intelligens med inriktning mot spel. Några av beslutsmetoderna, som *state machine* och *behavior tree,* baseras på traditionell matematisk logik, där en entitet antingen befinner sig ett specifikt tillstånd eller ej. Självklart går det att göra implementationer av de två nämnda strukturerna så att de befinner sig i flera tillstånd samtidigt. När en AI baseras på en struktur med binärt tänkande som resulterar i antingen eller, kommer övergångarna mellan olika tillstånd att bli mera skarpa och självklara. En spelare kommer med ganska stor sannolikhet att kunna avgöra vilka tillstånd en entitet i spelet befinner sig i. Med hjälp av oskarp logik går det att sudda ut övergångarna mellan olika tillstånd på ett helt annat sätt än med traditionell logik, eftersom oskarp logik baseras på graden av tillhörighet och inte antingen eller.

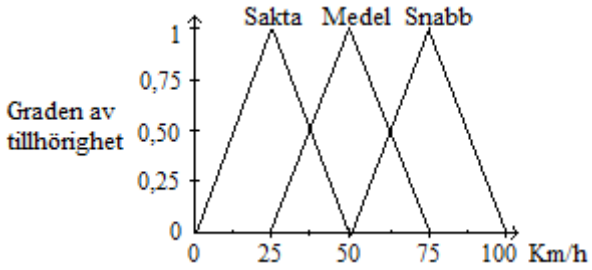
Vi föreställer oss ett rörligt objekt i ett digitalt spel. Vi delar in hastigheten i tre intervall under kategorierna: sakta, medel och snabb. Sedan representerar vi kategorierna för hastighet med hjälp av traditionell logik enligt figuren nedan.



Figur 12: Hastighet med skarp logik.

Figuren ovan ger oss ganska mycket information. Alla tre kategorier samt vilka intervall de är verksamma inom kan uttydas. Det finns två gränsland. Dessa är mellan sakta-medel och medel-snabb. Befinner sig ett objekt där, blir det svårt att avgöra hur snabbt bilen åker. Åker bilen sakta eller har en medelfart? Ett objekt som har hastigheten 49 är per definition sakta, men det är orimligt att en bil åker sakta när den nästan har medelfart. Objektet kan endast ha tillhörighet i ett av tillstånden.

Representerar figuren med hjälp av oskarp logik och väljer en triangulär funktion. Figuren nedan visar hur det skulle kunna se ut.



Figur 12: Hastighet med oskarp logik

Från figuren kan man istället utläsa mer information än i den förra. Ett objekt rör sig snabbare än medel, men betydlig långsammare än ett som färdas snabbt. Objektet kan ha en tillhörighet i både snabbt och medel som exempel. Med hjälp av det här synsättet kan man dra olika slutsatser för en spel-AI. Skillnaderna mellan dom olika tillstånden är inte lika skarpa. Om vi applicerar dom här två exemplen för skarp respektive oskarp logik i en spelsituation, kommer vi kunna dra olika slutsatser.

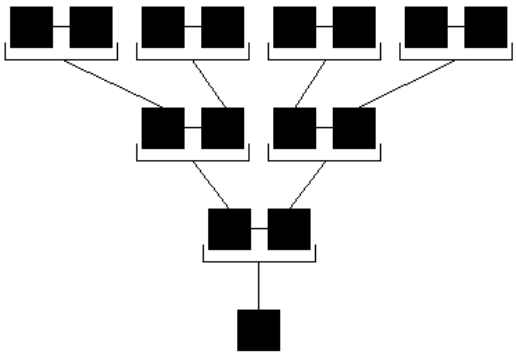
Låt oss anta att det finns en bil som är på väg in i en kurva. Om en beslutsteknik baseras på traditionell logik som *state machine* går det att avgöra när bilen skall bromsa eller accelerera. Om man istället använder oskarp logik går det att avgöra när något skall ske samt även hur mycket bilen behöver bromsa och accelerera för att klara en kurva. Det här är bara en typ av applicering som gör en stor skillnad för hur en AI upplevs. Med hjälp av oskarp logik går det att få en AI som beter sig mer oberäkneligt och det kan leda till en intressantare NPC.

4.3 Flervariabelbeslut

Oskarp logik baseras på hur mycket ett element tillhör en mängd och inte om den tillhör eller ej, vilket är fördelen och styrkan med oskarp logik. Med hjälp av oskarpa mängder går det att låta en NPC i ett digital spel basera sina beslut på många olika omständigheter och faktorer. Alla omständigheter kan komma att ändras under tidens gång, men det skapar inte några större problem. Innan omnämndes det i uppsatsen att regler med många ingående variabler kan leda till att många olika underregler behöver skapas och utvecklas. Bara för att man använder en massa faktorer behöver det inte leda till för många regler. Istället för att förlita sig på några regler som behandlar mycket indata används istället många regler med liten mängd indata.

Vi tittar på 3.1.2 där exemplet visar att ett AI-system kan tänkas använda ett specifikt vapen i olika situationer. Här beräknar vi ut en önskvärdhet som baseras på avståndet och hur mycket ammunition som finns kvar i vapnet. Exemplet innehåller nio stycken regler och med hjälp av dem går det att komma fram till väldigt många olika beslut. Här baseras beslutet gällande att skjuta på flera faktorer och det krävs inte så många regler, eftersom varje regel har två ingående värden. Dom här nio reglerna är en uppsättning för ett specifikt beslut.

Vi kan utveckla tanken på att det finns få regler och ingående värden. Det vi vet nu är att ett tillstånd baseras på regler. Om varje tillstånd har mindre antal ingående värden så kommer detta att resultera i ett mindre antal regler. Vi vet också att ett beslut kan baseras på en rad olika tillstånd och mängder. Skulle man låta ett beslut baseras på många olika mängder där varje regel för de ingående mängderna använder ett mindre antal ingångsvärden, då skulle man kunna få ett beslut att baseras på en rad olika faktorer. Vi använder exemplet under punkten 3.1.2 där önskvärdheten baseras på ammunition och skada. Resultatet av de reglerna skulle sedan kunna användas vidare till en ny uppsättning regler. Då skulle mängder kunna baseras på varandra och beslut skulle grunda sig i många olika faktorer. På det här sättet skulle en AI:s beslutstagande kunna bli i hög grad baserad på omgivningen utan att behöva allt för många regler. Figuren nedan visar hur mängder skulle kunna återanvändas i flera led innan resultatet man är ute efter beräknas och används.



Figur 13: Återanvändning av mängder

Detta skulle kunna användas till att göra så att större beslut för en NPC baseras på många olika faktorer i spelet. Låt oss säga att vi har en NPC som styr ett lag i ett RTS (Real Time Strategy) spel. Antingen låter vi det datorstyrda laget att fuska, dvs. att systemet matar laget med information om hur spelare agerar. Istället för att använda det tillvägagångssättet kan vi låta vår AI baseras på olika faktorer. Bara en sådan sak som att anfalla är en väldigt svår fråga som kräver mycket information innan beslut kan fattas. AI:n kommer ta i beräkning flera faktorer:

- Hur stor armé har jag?
- Har jag tillräckligt med resurser att kunna bygga en ny armé?
- Hittar jag till min fiende?
- Kan den här personen ha större arme än mig?

De här frågorna är väldigt relevanta för att kunna utföra en lyckad attack. Varje fråga baseras på olika faktorer, men tillsammans ger de en bra bild över hur lyckad en attack kan bli. Sådana här problem kommer dagens spel att ställas inför och antingen låter man NPC:n fuska eller så implementerar man en bättre lösning. Med hjälp av den här tankegången kan en AI lära sig och skaffa sig en egen uppfattning om hur saker och ting förhåller sig. Får en NPC olika grundförutsättningar så kommer besluten att ske olika sätt.

I digitala rollspel skulle den här tankegången kunna appliceras. När en spelare möter en annan karaktär som kontrolleras av en AI och inleder en dialog, behöver AI:n information om spelaren för att kunna skaffa sig en uppfattning om denne. För att avgöra om spelaren är farlig så kan AI:n avgöra det igenom flera olika faktorer som:

- Kroppsspråket på spelaren.
- Hur spelaren besvarar frågor.
- Vad spelaren har på sig eller inte har på sig.
- Hur spelaren beter sig i allmänhet.
- Om det finns något rykte om spelaren.

Det här är några faktorer som kommer att behövas för att skapa en bra bild över hur farlig en spelare är eller inte. Många olika faktorer kan spela en betydande roll för ett beslutstagande. Beslut kan innefatta många olika delar.

5 Slutsats

I den här uppsatsen ville jag fördjupa mig inom oskarp logik med inriktning emot artificiell intelligens för digitala spel. Idén var att sätta mig in i oskarp logik och sedan använda logiken som en grund för beslutstaganden i spel. Uppsatsen täcker endast in ren oskarp logik på ett teoretisk plan och har inte applicerats tillsammans med andra kända strukturer som exempelvis *FSM* och *beslutsträd* inom spel-AI.

Fördelen med oskarp logik är att den inte bygger på så kallade crisp-mängder, där ett element antingen tillhör en mängd eller inte. Mängderna inom oskarp logik bygger på graden av tillhörighet och det resoneras kring i vilken mån en mängd tillhör en annan. Istället för att säga att ett element befinner sig i mängden eller inte, säger man att det här elementet tillhör just den här mängden till en viss grad. Istället för att enbart vara välja mellan mätt och hungrig, går det att vara lite sugen. Det här tankegången är kärnan i oskarp logik.

AI som går att förutse vad den skall göra härnäst är en ointressant AI. En del spel idag lider av att deras datorstyrda enheter rör sig klumpigt och tar konstiga beslut. Många enklare strukturer baseras på traditionell logik där man antingen befinner sig i ett tillstånd eller inte. Resultatet kan bli att den AI-kontrollerade NPC:n rör sig konstigt och beter sig underligt, då skarvarna mellan tillstånden blir alltför uppenbara. Om man istället skulle göra övergångarna mellan tillstånden mjukare, finns det utrymme att göra intressantare AI. I exemplet under punkt 4.2 beskrivs hastighet på olika sätt. Utifrån det kan slutsatsen dras att en bil kör fortare än genomsnittet, men betydlig långsammare än en riktig fartsyndare. På det här sättet kan ett AI-system bli intressantare och mer realistisk.

Eftersom oskarp logik använder sig av graden av tillhörighet så går det att basera beslut på många olika faktorer. I ett RTS (Real Time Strategy) spel kan ett AI-system basera beslutet att anfalla ett annat lag på många olika faktorer. I exemplet under punkt 4.3 beskrivs olika förutsättningar som ett AI-system bör tänka över innan denne går till anfall. Skulle man mata ett AI-system med information som denne aldrig haft möjlighet till att skaffa själv, resulterar det för eller senare i att spelaren känner sig lurad. Skulle ett AI-system få för mycket info så skulle jag som spelare känna att mina egna beslut varken gör till eller från. Men om man baserar ett beslut på graden av tillhörighet för en mängd som har en tillhörighet till andra mängder, kan slutresultatet bli väldigt överraskande. Istället för en fördefinierad fiende har man nu ett AI-system som kan ta beslut på egen hand, utifrån sin egna uppfattning. Om ett AI-system utsätts för samma situation flera gånger och om besluten baseras på olika faktorer så kommer inte utfallet bli densamma varje enskild gång.

Självklart finns det nackdelar med oskarp logik. Antag att vi har en regel med många ingående variabler och varje variabel har ett eller flera tillstånd. Under dessa förutsättningar kommer det att behöva skapas väldigt många olika regler för att hantera varje möjligt utfall. Många av nackdelarna med oskarp logik uppstår på grund av felaktig användning. Istället för att förlita sig på många variabler som ingår i en och samma regel, kan man istället ha flera uppsättningar regler som förlitar sig på få antal variabler. På det här sättet kommer inte utvecklaren att behöva skapa lika många regler rent matematiskt, utan kan basera sitt beslut på många olika faktorer. Helheten kommer bli att ett AI-system har många olika faktorer att basera sitt beslut utifrån, där varje faktor inte behöver vara överkomplicerad.

6 Diskussion

Den här uppsatsen går igenom oskarp logik väldigt grundläggande och gör sedan en applicering inom artificiell intelligens för digitala spel. En stor anledning till att oskarp logik endast behandlas på en grundläggande nivå är för att det inte går att kräva att läsaren har förkunskaper inom diskret matematik, då diskret matematik inte ligger inom ramen för rapportens huvudämne speldesign. Det hela medför problem eftersom nivån på genomgången av logiken blir väldigt elementär och delar av ämnet inte kommer att behandlas. Resultatet ger inte en helt rättvis bild över själva logiken, men en tillräcklig bild för att förstå konceptet.

Appliceringen av oskarp logik i den här uppsatsen blandas inte med andra vanligt förekommande tekniker inom AI, som *Finit State Machine* eller *Behavior Trees*. Det går att få stöd av logiken inom dom här teknikerna och det är inget som nyttjas i den här uppsatsen. Bilden av hur effektivt oskarp logik kan vara blir inte helt rättvis. I teorin kan en logik fungera väldigt bra ensam. Men då ingen praktisk implementation har skett, så blir slutsatsen endast baserad på teorier och inte på praktisk erfarenhet.

Då skarp logik är både en ung gren inom matematiken och inom artificiell intelligens för digitala spel, har varken logiken eller dess applicering inom AI ännu hittat sin rätta plats. Den här uppsatsen kan fungera som en introduktion till denna teknik. Det finns många möjliga öppningar för vidare forskning och undersökning. Några naturliga steg om man utgår ifrån den här uppsatsen skulle vara följande:

- Använd oskarp logik tillsammans med vanligt förekommande tekniker inom artificiell intelligens för digitala spel. Några tekniker kan till exempel vara: neurala nätverk, finite state machine (FSM) eller behavior tree. För att testa hur logikens fördelar kan speglas inom andra tekniker görs en digital implementation.

- Utföra ett digitalt test där man ställer en vanligt förekommande teknik mot en som är blandad med oskarp logik, för att kunna avgöra om oskarp logik för med sig nämnvärda fördelar. Det skulle vara intressant att se skillnaden mellan en AI baserad på oskarp logik och en som inte är det.

- Det skulle var väldigt intressant att se andra appliceringsområden som använder sig av oskarp logik och en blandning av AI för digitala spel. Denna implementation skulle inte enbart kunna appliceras på själva logiken utan även strukturen som *FSM* eller *Beslutsträd*.

Självklart finns det många flerområden som oskarp logik kan tänkas att fungera inom. Förslagen på vidare forskning som anges ovan är mina egna tankar och åsikter. Man skall inte döma ut en logik förrän den har testats i många olika applikationer. Oskarp logik används inom många andra områden än spel.

Referenser

Adams, Ernest & Dormans Joris 2012. *Game Mechanics: Advanced Game Design*. Berkeley: New Riders Games

Bakkes, Sander C.J & Spronck, Pieter H.M & van den Herik, Jaap 2009. Opponent modelling for case-based adaptive game AI. I: *Entertainment Computing 1*. S. 27 - 37

Bojadziev, George & Bojadziev, Maria 2007. *Fuzzy Logic For Business, Financed Management*. River Edge: World Scientific

Buckland, Mat 2005. *Programming Game AI by Example*. Wordware Plano: Publishing, Inc.

Champandard, Alex J 2003. *AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors*. New Jersey: New Riders Publishing

De Silva, C.W 2011. Zadeh-Mac-Farlane-Jamshidi theorems on decoupling of a fuzzy rule based. I: *Scientia Iranica 18*. S. 611 - 616

Demico, Rovert V & Klir, George J 2003. *Fuzzy Logic in Geology*. San Diego: Academic Press

Dobrev, Dimiter 2005. A Definition of Artificial Intelligence. I: *Mathematica Balkanica, New Series, Vol. 19, 2005, Fasc 1-2* S. 67-74

Eriksson, Kimmo & Gavel, Hillevi 2010. *Diskret matematik och diskreta modeller*. Malmö: Studentlitteratur

Jo Son, Myeong & Kim, Tae-wan 2012. Torpedo evasion simulation of underwater vehicle using fuzzy-logic-based tactical decision making in script tactics manager. I: *Expert System with Applications, Volym 39, Issue 9* S. 7995-8012.

Kose, Utku 2012. Developing a Fuzzy Logic Based Game System. I: *Computer Technology and Application 3*. S. 510 - 517.

Koshy, Thomas 2003. *Discrete mathematics with Applications*. Burlington: Academic Press

Lee Mitchell, Briar 2012. *Game Design Essentials*. Indianapolis: John Willey & Sons Inc.

Ma, J & Chen, S & Xu, Y 2006. Fuzzy logic from the viewpoint of machine intelligence. I: *Fuzzy Sets and Systems, Vol. 157, Issue 5*. S. 628 - 634

Millington, Ian & Funge, John 2009. *Artificial intelligence for Games*. San Francisco: Elsevier

Myeong-Jo, Son & Tae-wan, Kim 2012. Torpedo evasion simulation of underwater vehicle using fuzzy-logic-based tactical decision making in script tactics manager. I: *Expert System with Application 39*. S. 7995-8012.

Nilson, Nils J 2010. *The Quest for Artificial Intelligence*. Cambridge University press

Saffidine, Abdallah & Finnsson, Hilmar & Buro, Micheal 2012. Alpha-Beta Prunning for Games with Simultaneous Moves. I: *AAAI*. S. 556 - 562

Schwab, Brian 2009. *AI Game Engine Programming*. Boston: Course Technology

Shaout, Adnan & King, Brady & Reisner, Luke 2006. Real-Time Game Design of Pac-Man Using Fuzzy Logic. I: *The International Arab Journal of Information Technology*. S. 315 - 325

Sreekumar, Acharjya D.P 2009. *Fundemental Approach to Discrete Mathematics*. Daryaganj: New Age International

Rollings, Andrew & Adams, Ernest 2003. *Andrew Rollings and Ernest Adams on Game Design*. New Riders Publishing

Russell, Stuart & Norvig, Peter 1995. *Artificial Intelligence A Modern Approach*. New Jersey: Prentice Hall, Inc.